



Python Tutorial

...

Hyeyoung Ryu (2016195045)

Installation for Python Tutorial

Installing Python 3

Installing Python 3 - Mac Users

1. Open terminal
2. Install Homebrew
 - `$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`
3. Install Python
 - `$ brew install python`

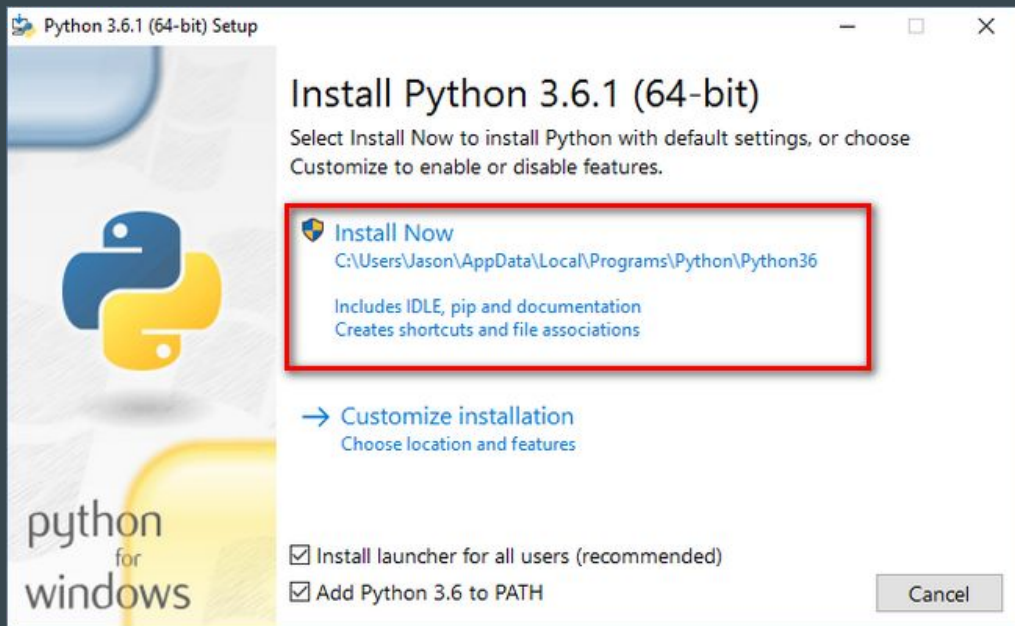
Installing Python 3 - Windows Users

1. Go to the website written below
 - <https://www.python.org/downloads/windows/>
2. Install the Windows x86-64 executable installer

- [Python 3.6.1 - 2017-03-21](#)
 - Download [Windows x86 web-based installer](#)
 - Download [Windows x86 executable installer](#)
 - Download [Windows x86 embeddable zip file](#)
 - Download [Windows x86-64 web-based installer](#)
 - Download [Windows x86-64 executable installer](#)
 - Download [Windows x86-64 embeddable zip file](#)
 - Download [Windows help file](#)

Installing Python - Windows Users

- ***WARNING*** Make sure to click “Add Python 3.6 to PATH” before clicking the “Install Now” button



Python Development Environments



Python IDLE



PyCharm



Jupyter Notebook

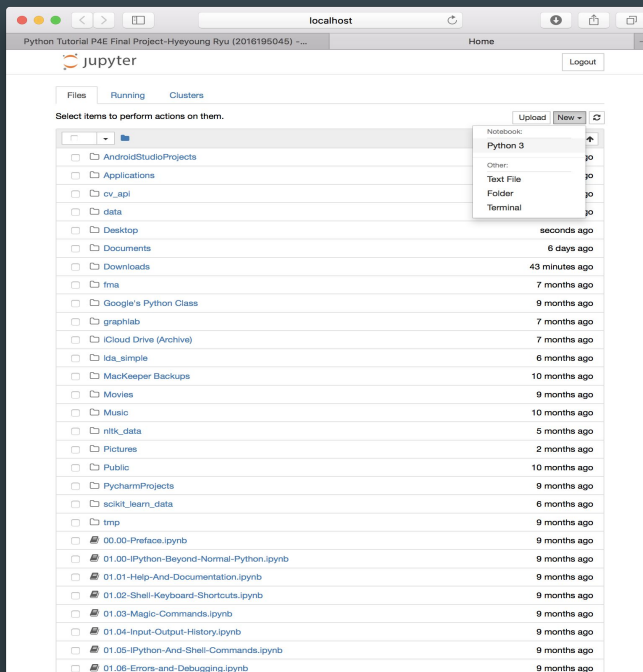
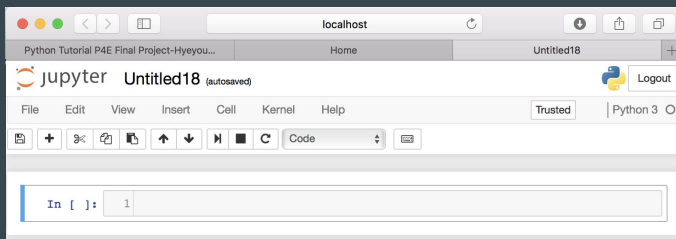
Installing Jupyter Notebook

Installing Jupyter Notebook

1. For Mac users, open terminal and for Windows users, open command prompt
2. Type the following
 - `python3 -m pip install -- upgrade pip`
 - `python3 -m pip install jupyter`
3. After having installed jupyter notebook, type “jupyter notebook” into terminal or command prompt to use it

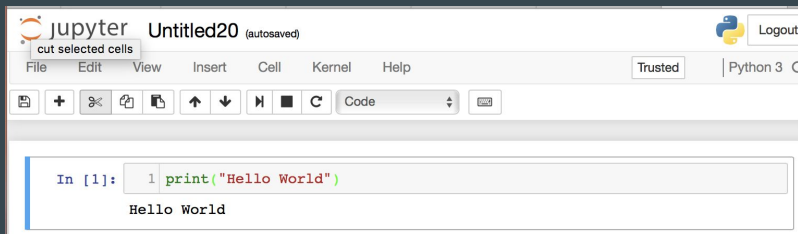
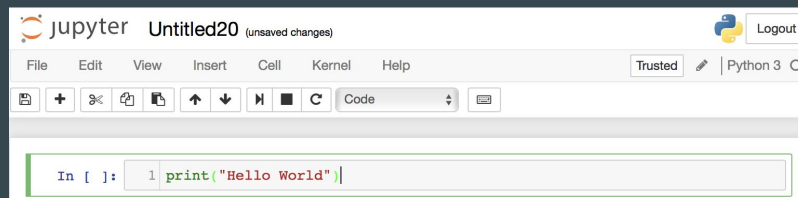
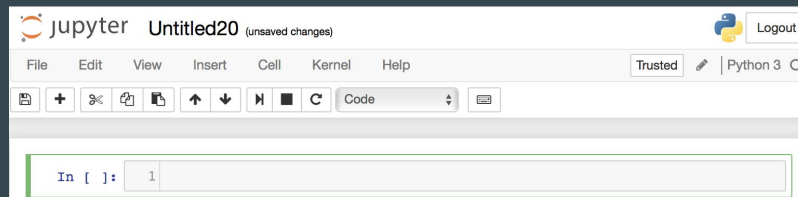
Using Jupyter Notebook

1. Click “New”
2. Click “Python 3”
3. New notebook will be made



Using Jupyter Notebook

1. Write the code on the green boxed section
2. To run the code, click shift and enter at the same time
3. The results will be shown below



Before We Begin Python

Why Use Python?

- Easy and intuitive language
- Innumerable resources
- Easy for beginners
- Relatively fast development speed



Print () Function

- Built-in function
- Prints out values to the screen
 - `print ("Tom Hiddleston is the sexiest man alive.")`
- Can pass several values
 - `print ("Black", " ", "Panther")`

Variables

- Location in memory used to store some value
- Unique names are given to variables to differentiate between different memory locations
- Assignment statement
 - Creates new variables and gives them values
 - “=” is used to assign a value to a variable
- `series = “How I Met Your Mother”`
- `print (“The revival of the TV series ‘Friends’ was ”, series, “.”)`

Comments

- Line of text that the Python interpreter completely ignores when executing the code
- Use hash symbol (#) to indicate comments
 - Can only write a single physical line of comments with #
- Use “""" """” to write many lines of comments
- Need to write comments for the code to be understandable to you and other programmers
- #Assign string value to the variable series
- series = "How I Met Your Mother"

Path

- Directory
- Works like an address
- E.x. your computer's "Downloads" folder address
 - Window Users: "C:/Users/username/Downloads"
 - Mac Users: "/Users/username/Downloads"

Let's Start Python!

Instruction: For Every
Turquoise Line (Codes),
Run It In Jupyter Notebook

1. Data Types

Data Types

- Think of the data types as the **alphabet** of Python
- Numbers
- Strings
- Boolean
- Lists
- Tuples
- Dictionary
- Set

Numbers

- Integers
 - Round numbers without a decimal point
 - E.x. 2, 4, -15
- Floating Numbers
 - Real numbers
 - May have numbers after decimal points
 - E.x. 2.7, 4.123, -12.9
- Complex Numbers
 - Numbers in the form $a + bi$
 - a, b = real numbers
 - i = solution of " $x^2 = -1$ "
 - E.x. $2 + 5i$

Numbers - Basic Math

- Addition: +
 - `print (7 + 8)`
- Subtraction: -
 - `print (5 - 1)`
- Multiplication: *
 - `print(15 * 10)`
- Division: /
 - `print (20 / 5)`
- Modulo: %
 - Returns the remainder
 - `print (56 % 9)`
- Exponentiation: **
 - `print (2 ** 3)`

Strings

- Sequence of letters
- Indicating strings
 - Single (' ')
 - Double (" ")
 - Three of each quotation mark (" " " " , ' ' ' ')
- `hello = "Hello World!"`
- `print (hello)`

Strings - Addition

- Let's say you want to write the sentence "My name is Tom Cruise."
- `first_name = "Tom"`
- `last_name = "Cruise"`
- `print ("My name is " + first_name + " " + last_name + ".")`

Strings - Multiplication

- To make your wish come true, you have to say “Bibidi Babidi Bu” twice.
- `cinderella = “Bibidi Babidi Bu”`
- `print (cinderella * 2)`

Strings - Indexing

- Python starts from 0!
 - If you want to know the first letter of the word, you have to get index 0, not index 1
- Let's say that you want to know the first letter of the word "supercalifragilisticexpialidocious"
 - `poppins = "supercalifragilisticexpialidocious"`
 - `print (poppins[0])`
- Let's say that you want to know the third letter of the word
 - `print (poppins[2])`
- Let's say that you want to know the third to last letter of the word
 - ***HINT*** Index of last letter of the word is "-1"
 - `print(poppins[-3])`

Strings - Indexing

- Let's spell "taxi" from "supercalifragilisticexpialidocious"
 - `print (poppins[17] + poppins[6] + poppins[-13] + poppins[-4])`
- Let's spell "super" from "supercalifragilisticexpialidocious"
 - `print (poppins[0:4])`

Strings - Format

- Using the `format()` method we can add in variable values and generally format our strings
 - By using `{ }`, you can assign which part of the string will be filled later
 - By using `format()` function, you can assign the value that goes into `{ }`
- `my_name = "{0} {1}".format("Chandler", "Bing")`
- `print (my_name)`

Strings - Format

- % symbol creates a placeholder
- Conversion type
 - Whatever character follows % indicates the type of value put into the placeholder
 - %s = string
- Need another % after the string has closed
 - % will be followed by the values to insert
 - In the case of one value, you can just put it there
 - `print ("Guardians of the Galaxy has %s members" % "four")`
 - If you are inserting more than one value, they must be enclosed in a tuple
 - `print ("%s bottles of soju makes Kevin %s." % (7, "go night night"))`

Strings - Functions

- Count() function
 - Returns the number of character in the string
- `a = "Python is fun"`
- `a.count("n")`

Strings - Functions

- Find () function
 - Returns the value of first position
 - If value does not exist, return -1
 - `print(a.find("n"))`
 - `print(a.find("z"))`
- Index () function
 - Returns the index number
 - `print(a.index("n"))`

Strings - Functions

- Upper () function
 - Capitalize string
 - `print(a.upper())`
- Lower () function
 - Lowercase string
 - `print(a.lower())`

Strings - Functions

- Strip () function
 - Deleting spaces for both sides of string
 - `b = " Excessive spaces need to be stripped. "`
 - `print(b.strip())`
- Lstrip () function
 - Delete spaces on left side of string
 - `print(b.lstrip())`
- Rstrip () function
 - Delete spaces on right side of string
 - `print(b.rstrip())`

Strings - Functions

- Split () function
 - Splits string into words
 - `sentence = "I/hate/working/night/and/day"`
 - `print(sentence.split("/"))`

Strings - Functions

- Replace () function
 - Replaces part of the string to what you want to change it to
 - `a.replace("fun", "horrible")`

Boolean

- Boolean expression is evaluated as either True or False
- Relational operators

Symbols	Meanings
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Boolean

- Logical operators
 - And (&)
 - True and True ➡ True
 - True and False ➡ False
 - False and False ➡ False
 - Or (|)
 - True or True ➡ True
 - True or False ➡ True
 - False or False ➡ False
 - Not
 - Not True ➡ False
 - Not False ➡ True

Lists

- Ordered collection of objects that can contain any data type
- Can be defined using brackets []
- Element
 - Each data in the list
- `alcohol = ["Jose Cuervo", "Bombay Sapphire", "Baileys", "KGB", "Bernini"]`
- `print(alcohol)`

Lists - Index

- To find a specific element in the list
 - Type in the index number inside the brackets of “list name[]”
 - `print(beer[0])`
- Indexing/slicing
 - Get the items after the first element from the list beer
 - `print(beer[1:])`
 - Get the third and fourth element of the list beer
 - `print(beer[2:4])`
 - Get until the third element from the list beer
 - `print(beer[:3])`

Lists - +

- Two different lists can be combined into one list with “+”
- `romance = [“10 Things I Hate About You”, “Love Actually”, “Pretty Woman”, “50 First Dates”, “500 Days of Summer”]`
- `horror = [“The Conjuring”, “Annabelle”, “It”, “The Shining”, “Scream”, “Paranormal Activity”]`
- `movies = romance + horror`
- `print (movies)`

Lists - Functions

- Len () function
 - Returns the number of elements in a list
 - `print(len(romance))`
- Count () function
 - Counts how many times the element is repeated in the list
 - `list = [2, 3, 4, 2, 5, 2, 9]`
 - `print(list.count(2))`

Lists - Functions

- Append () function
 - Adds an element to the list
 - `romance.append("When Harry Met Sally")`
 - `print(romance)`
- Insert () function
 - Adds element into the list if the element is not in the list
 - `list = [2, 3, 4, 2, 5, 2, 9]`
 - `list.insert(3, 10)`
 - `print(list)`

Lists - Functions

- Extend () function
 - Extends the list
 - `a = [1,2,3,4,5]`
 - `a.extend(["hello","5",5])`
 - `print(a)`
- Remove () function
 - Removes an element from the list
 - `horror.remove("The Shining")`
 - `print(horror)`

Lists - Functions

- Sort () function
 - Sorts elements in the list
 - `c = [6,4,8,"e",3,"b","c"]`
 - `c.sort()`
 - `print(c)`
- Reverse () function
 - Sorts elements in the list in reverse order
 - `a = ["a","c","b"]`
 - `a.reverse()`
 - `print(a)`

Tuples

- Immutable
 - Data cannot be changed once it has been generated
- When tuple has only one value, a comma must be followed
- `tuple1 = (1,)`
- `print(type(tuple1))`
- Bracket can be skipped when assigning variables with tuple
- `tuple2 = (1, 2, 3)`
- `print(type(tuple2))`
- `tuple3 = 1, 2, 3`
- `print(type(tuple3))`

Tuples - Index

- Tuples indexing is similar to lists
- `tuple4 = (1, 2, 7, 4, "hello", "bye", 1, "7")`
- Get the fifth element in the tuple
 - `print(tuple4[4])`
- Get from the third element in the tuple
 - `print(tuple4[2:])`

Tuples - Addition and Multiplication

- Addition
 - `tuple5 = (1,2,5)`
 - `tuple6 = (4,6,7)`
 - `tuple7 = tuple5 + tuple6`
 - `print(tuple7)`
- Multiplication
 - `print(tuple5*2)`

Dictionary

- Map between key set and value set
- Set of pairs of (key, value)
- Key must be immutable
- Combination of curly brackets { } and colons :
 - Curly brackets define the beginning and end of a dictionary
 - Colons indicate key-value pairs
- ***CAUTION*** Dictionary key is a unique value
 - Same key cannot be used again
 - List cannot be used in the key
- `personal_Info = {"name":"Sarah", "email":"hocuspocus@gmail.com", "birthday":"19001031"}`

Dictionary - Functions

- Keys () function
 - Retrieves only keys from the dictionary
- Values () function
 - Retrieves only values from the dictionary
- Items () function
 - Retrieves keys and values pairs from the dictionary
- Get () function
 - Gets values with keys

Dictionary - Functions Example

- `idols = {"Twice":
["Jihyo","Nayeon","Jungyeon","Momo","Sana","Mina","Dahyun","Chaeyoung","Tzuyu"], "Blackpink": ["Jisu","Jennie","Rose","Lisa"], "Bigbang":
["GD","TOP","Taeyang","Daesung","Seungri"], "BTS":
["RM","Suga","Jin","J-Hope","Jimin","V","Jungkook"]}`
- Retrieve the keys from idols dictionary
 - `print(idols.keys())`
- Retrieve the values from idols dictionary
 - `print(idols.values())`
- Retrieve keys and values pairs from the idols dictionary
 - `print(idols.items())`
- Get values with keys for "BTS" from the idols dictionary
 - `print(idols.get("BTS"))`

Sets

- Made with `set()` function
- Does not allow repetitive values
- Unordered
 - Cannot be indexed or sliced
- `s1 = set([1,2,3])`
- `s2 = set("This is set")`
- `print (s1, s2)`

Sets - Index

- To index or slice sets, convert to tuples or lists
- `l2 = list(s2)`
- `print(l2)`
- Find the fifth element in l2
- `print(l2[4])`

Type Conversion

- Built-in functions for type conversion
 - Int ()
 - `a = int(3.14)`
 - `print(type(a))`
 - Float ()
 - `b = float("123.4")`
 - `print(type(b))`
 - Str ()
 - `c = str(25)`
 - `print(type(c))`
 - Round ()
 - `d = round(1.23)`
 - `print(type(d))`

HELPFUL NOTE
Type () function returns
the type of the given
object

Sets - Intersection

- `a = set([1,2,5,7,8,9])`
- `b = set([2,6,3,8,4])`
- `print(a & b)`
- `print(a.intersection(b))`

Sets - Union

- `a = set([1,2,5,7,8,9])`
- `b = set([2,6,3,8,4])`
- `print(a | b)`
- `print(a.union(b))`

Sets - Subtraction

- `a = set([1,2,5,7,8,9])`
- `b = set([2,6,3,8,4])`
- `print(a - b)`
- `print(a.difference(b))`

Sets - Functions

- Add () function
 - Adding a value to the set
 - `s1 = set([1,4,6,7])`
 - `s1.add(9)`
 - `print(s1)`

Sets - Functions

- Update () function
 - Adding multiple values to the set
 - `s2 = set([5,8,3])`
 - `s2.update([1,4,5,6])`
 - `print(s2)`

Sets - Functions

- Remove () function
 - Removing a value from the set
 - `s3 = set([2,5,8,1])`
 - `s3.remove(2)`
 - `print(s3)`

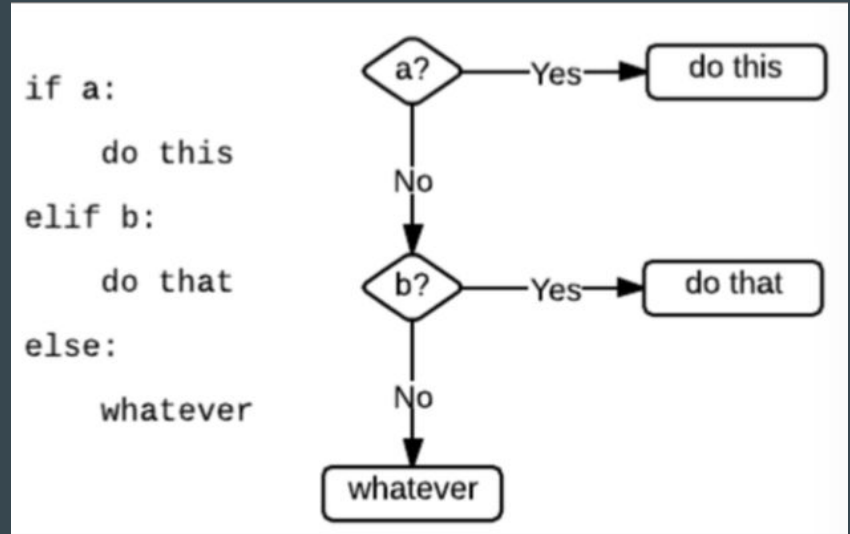
2. Control Statements

Control Statements

- If conditional statement
 - If
 - Elif
 - Else
- Loop statement
 - For
 - While

If Conditional Statement

- Statement that may or may not be executed depending on certain conditions that the code specifies
- Condition part is when Boolean data types and logical operators are used frequently



If Conditional Statement Exercise

- Let's say that you are 15 years old, but want to watch Deadpool 2 which is a R-rated movie.
- `my_age = 15`
- `if my_age >= 19:`
 - `print ("Go ahead and chill with Deadpool!")`
- `elif 15 <= my_age < 19:`
 - `print ("You may watch, but you have to watch with a guardian.")`
- `else:`
 - `print("Go home honey.")`

For Loop

- For is a loop statement used for going through values or iterating over any sequences (tuple, list, string)
- `list = [0, 3, 6, 9]`
- `for item in list:`
 - `print (item)`

While Loop

- While loop statements are repeated while a certain condition is met
- When condition becomes false, the loop stops

- `audition = 0`

```
while audition < 10:
```

```
    audition = audition +1
```

```
    print("Jennifer Aniston had to audition %d times." % audition)
```

```
if audition == 10:
```

```
    print("Jennifer Aniston got to play Rachel on Friends.")
```

Break, Continue, Pass

- Break
 - When triggered, breaks out of the loop
- Continue
 - When triggered, skips part of the loop
- Pass
 - Runs entire loop

Break

- `number = 0`
- `for number in range (10):`
 - `number = number + 1`
 - `if number == 5:`
 - `break`
 - `print ("Number is " + str(number))`

Continue

- `number = 0`
- `for number in range (10):`
 - `number = number + 1`
 - `if number == 5:`
 - `continue`
 - `print ("Number is " + str(number))`

Pass

- `number = 0`
- `for number in range (10):`
 - `number = number + 1`
 - `if number == 5:`
 - `pass`
 - `print ("Number is " + str(number))`

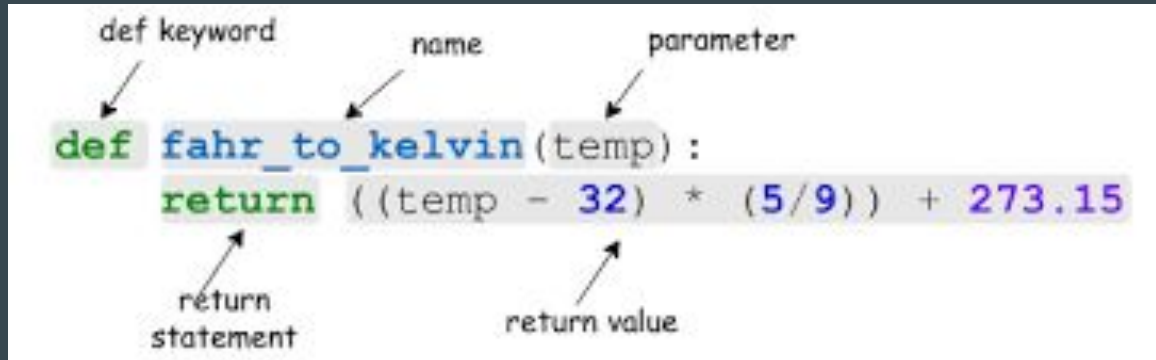
3. Functions

Indentation

- Indent using the “Tab” key
- 4 spaces = 1 indentation
- Code block
 - Group of statements that perform a task together
 - E.x. functions
 - Indented statements + preceding header
 - Begins with a header that is followed by one or more statements that are indented with respect to the header

Functions

- Sequence of statements that performs a computation
- Takes an argument
- Returns a result
- Built-in or defined by user
- Call the function by name and parameter(s)



```
def fahr_to_kelvin(temp):  
    return ((temp - 32) * (5/9)) + 273.15
```

The diagram illustrates the components of a Python function definition. Arrows point from labels to specific parts of the code: 'def keyword' points to 'def', 'name' points to 'fahr_to_kelvin', 'parameter' points to '(temp)', 'return statement' points to 'return', and 'return value' points to the expression '((temp - 32) * (5/9)) + 273.15'.

Built-In Functions

- Can be called without having to newly define them
- Ex. `input ()` function
 - Receives user input and returns the input
 - `Input (prompt)`
 - Prompt is a string and the message that appears to the user when inputting values
 - `game_of_thrones = input("Who will sit on the throne in the end? ")`
 - `game_of_thrones`

Defining Functions

- Define a function with keyword `def` and parameters
 - Parameters are optional
- Header ends with a colon
- Body is indented
- Docstring
 - Describes what the function does
 - Write inside `“““ ”””`
- `def game_of_thrones_greeting (person):`
 - `“““This function greets people in a game of thrones way.”””`
 - `print (“Winter is coming. ” + “Be prepared ” + person + “.”)`
- `game_of_thrones_greeting (“Sansa Stark”)`

Return Statement

- Allows you to return to a result from a function
- Used to exit a function and go back to the place from where it was called
- Can call a function inside another function
- `def add_ten(value):`
 - `"""This function adds ten to the value."""`
 - `result = value + 10`
 - `return result`
- `add_ten(21)`

4. Modules

Installing Modules

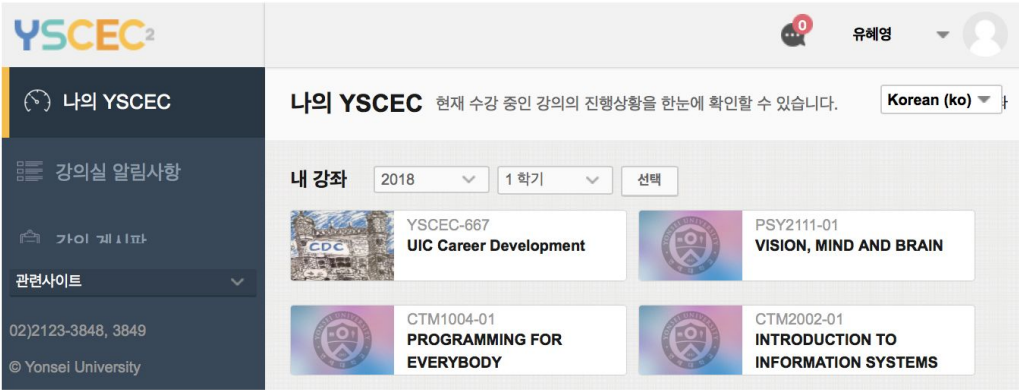
- Windows users
 - Use command prompt
- Mac users
 - Use terminal
- Call pip to install required python modules
 - In command prompt or terminal type either of the following
 - `pip3 install module_name`
 - `python3 -m pip install module_name`

Using Modules - IFrame Example

- Let's open up yscec from jupyter notebook using IFrame
- `from IPython.display import IFrame`
- `IFrame("http://yscec.yonsei.ac.kr", width=800, height=300)`

```
In [8]: 1 from IPython.display import IFrame
        2 IFrame("http://yscec.yonsei.ac.kr", width=800, height=300)
```

Out[8]:



The screenshot displays the YSCEC2 web application. The top header includes the YSCEC2 logo, a notification icon, the user name '유혜영', and a profile icon. The main content area is titled '나의 YSCEC' and includes a description: '현재 수강 중인 강의의 진행상황을 한눈에 확인할 수 있습니다.' (You can check the progress of the courses you are currently taking at a glance). There is a language dropdown menu set to 'Korean (ko)'. Below this, there is a section for '내 강좌' (My Courses) with filters for '2018' and '1 학기' (1 semester). Four course cards are displayed:

- YSCEC-667 UIC Career Development
- PSY2111-01 VISION, MIND AND BRAIN
- CTM1004-01 PROGRAMMING FOR EVERYBODY
- CTM2002-01 INTRODUCTION TO INFORMATION SYSTEMS

5. Class

Class

- Data structure that can be reused and extended according to object-oriented programming paradigm
- Class = cookie frame
- Object = cookie
- `__init__`
 - Constructs an instance of a class

Class Example - Employees and Salary

Making class

```
class Employee:
    """Common basic information of all
    employees"""
    empNum = 0
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empNum += 1
    def show_empNum(self):
        print ("Total Employee Number is
        {0}".format(Employee.empNum))
    def show_empInfo(self):
        print ("Name: ", self.name, ", Salary: ",
        self.salary)
```

Class Example - Employees and Salary

Shaping objects

```
emp1 = Employee("Angie","5000")  
emp2 = Employee("Mike","4000")  
emp3 = Employee("Brian","3000")
```

Class Example - Employees and Salary

Using functions defined in the
class

```
emp2.show_empInfo()
```

```
emp3.show_empNum()
```

**Let's Take It To The Next
Level!**

**Exercise 1: Add all the even numbers that
exist from 0 to 100**

Step 1

From 0 to 100, the numbers
increase by adding one to the
previous number

```
num = 0
```

```
while num <= 100:
```

```
    num = num + 1
```

Step 2

Get the even numbers from the numbers from 0 to 100

```
num = 0
```

```
while num <= 100:
```

```
    num = num + 1
```

```
    if num % 2 == 0:
```

Step 3

Add all the even numbers and
save it into a variable

```
sum = 0
```

```
num = 0
```

```
while num <= 100:
```

```
    num = num + 1
```

```
    if num % 2 == 0:
```

```
        sum = sum + num
```

Step 4

Check the answer by printing the sum variable

```
sum = 0
```

```
num = 0
```

```
while num <= 100:
```

```
    num = num + 1
```

```
    if num % 2 == 0:
```

```
        sum = sum + num
```

```
print(sum)
```

Exercise 2: Create a Fibonacci Sequence

Exercise 2 Instructions

- You are given the first three numbers of the Fibonacci sequence
 - `F = [0, 1, 1]`
- Create a for loop to determine the next 20 numbers of the Fibonacci sequence
- Print F with the final 23 numbers
- Use `F.append()` to add a new Fibonacci value to the end of the list F

Step 1

Create a list with the first three
numbers of the Fibonacci
sequence

$F = [0, 1, 1]$

Step 2

Create a for loop to find the numbers in the Fibonacci series in the range (3, 23)

```
F = [0, 1, 1]
```

```
for i in range (3, 23):
```

```
    F.append(F[-1] + F[-2])
```

Step 3

Print the Fibonacci numbers on
the list “F”

```
F = [0, 1, 1]
```

```
for i in range(3, 23):
```

```
    F.append(F[-1] + F[-2])
```

```
print (F)
```

Good Job:D

**Now You Know The Basics
Of Python!**

Thirsty For More?

Check Out These Websites

- <https://www.codecademy.com/learn/learn-python>
- <https://www.udacity.com/course/programming-foundations-with-python--ud036>
- <https://teamtreehouse.com/learn/python>
- <https://www.programiz.com/python-programming>

Python Tutorial Exit ()